Lec : 7

Advanced Optimization ALgorithms:

- These algorithms are able to get Logistic regression to run much more quickly than it's possible with gradient descent (No Need to manually pick α) (Advantage).
- They also scale much better to very Large machine learning problems, such as IF we had a very Large number of Features. (Advantage)
- More Complex (disadvantage)

→ In Optimization Algorithms we use ready code (ready Functions) In MatLab called "fminunc" This code can be used with Linear regression, Logistic regression .... this means that optimization algorithms don't care about the Algorithm type as it only need the code that can compute $J(\theta)$ and $\frac{\partial}{\partial \theta_i} J(\theta)$

Optimization ALgorithms:

- Gradient Descent
- Conjugate Gradient
- BFGS
- L-BFGS

Fminunc attempts to Find a minimum of a scalar function of several Variables Starting at an initial estimate.

```
[optTheta, FunctionVal, exitFlag] = fminunc(@CostFunction, initialTheta, options);
```

(optTheta: return; FunctionVal: return; exitFlag: return; CostFunction: need; initialTheta: need; options: need)

The Function arguments:

1. The objective Function to be minimized (the Cost Function)
2. The initial Values of the Function variables (parameters θ)
3. The Optimization options specified in Options.

The Function return:

1. The optimal values of the Function variables (parameters θ) at which the local minimum of the Function.
2. The value of the objective Function at the optimal values of the parameters.
3. A value exitFlag that describes the exit condition.

Example:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad , \quad J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

we need to get $\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$ & $\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

Code

```
Options = optimest ('GradObj', 'on', 'MaxIter', '100');
InitialTheta = Zeros (2, 1);
[optTheta, FunctionVal, exitFlag] ...
        = fminunc (@CostFunction, InitialTheta, Options);
```

```
Function [jVal, gradient] = CostFunction (theta)
    jVal = (theta(1) - 5)^2 + ... (theta(2) - 5)^2 ;
gradient = Zeros(2, 1);
gradient(1) = 2* (theta(1) - 5);
gradient(2) = 2 * (theta(2) - 5);
```

Code In General

→ Theta = $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$, 1st Step Calculate CostFunction

Function [jVal, gradient] = CostFunction

jVal = [Code to Compute $J(\theta)$];

gradient (1) = [Code to Compute $\frac{\partial}{\partial \theta_0} J(\theta)$];

gradient (2) = [Code to Compute $\frac{\partial}{\partial \theta_1} J(\theta)$];

⋮

gradient (n+1) = [Code to Compute $\frac{\partial}{\partial \theta_n} J(\theta)$];

2nd Step determine the optimization Options

Options = optimset ('GradObj', 'on', 'MaxIter', '100');

This Function determine the optimization options, the most important options are The state of the obtimization objective (we choose gradient descent as an obtimization objective) we make the state 'on' to make it able to be used. AND the maximum number of iterations
For Simplification, we always make GradObj 'on' and MaxIter '100'

3rd Step Initial Values of θ "zeros"

InitialTheta = Zeros (number of θ (n) , 1);

number of raw

↓ number of column

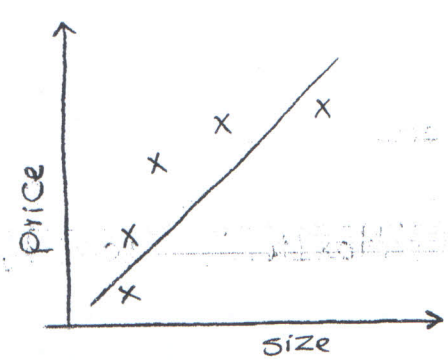
4th Step get Fminunc Function

```
[optTheta, FunctionVal, exitFlag] =
        = Fminunc(@CostFunction, Initial Theta, Options);
```

→ This Function takes 1. CostFunction 2. Initial Theta 3. Options and gives Optimal Value of theta that gives the optimal Solution, Optimal Value For CostFunction (equals to Zero at perfect solution) AND returns exitFlag which Value equals to 1 when the Code Finishes running.

Regularization

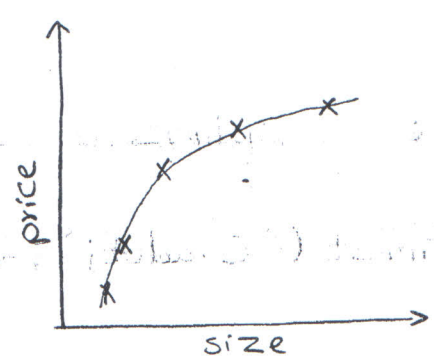
Example: Linear regression (housing prices)



$h_\theta(x) = \theta_0 + \theta_1 x$

Under Fitting

here, we don't Fit the data at all.

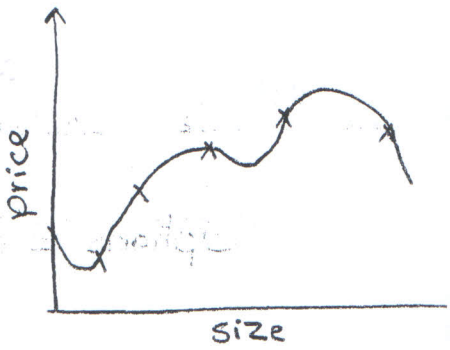


$h_\theta(x) = \theta_0 + \theta_1 X + \theta_2 X^2$

Just Right

here, we just Fit this data problem we may not be able to get the right answer For another data.

(The Best Way For Fitting Data)

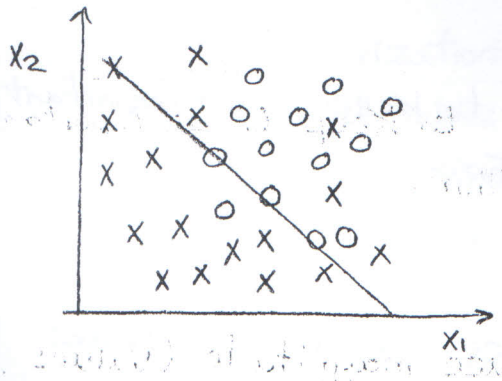


$h_\theta(x) = \theta_0 + \theta_1 X + \theta_2 X^2 + \theta_3 X^3 + \theta_4 X^4$

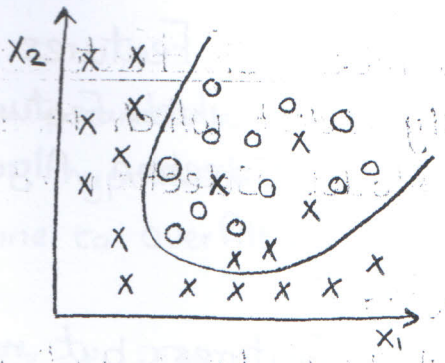
Over Fitting

here, we get the best Fitting For this data b we can't Fit any othe data!

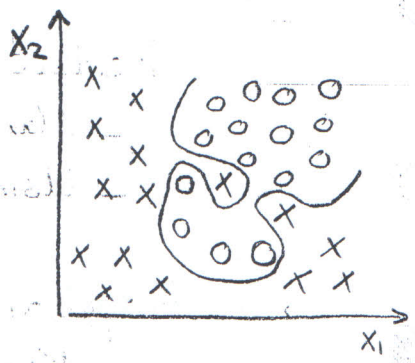
Example : Logistic regression



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \cdots)$

where g : Sigmoid Function.

Under Fitting | Just Right | Over Filling

Overfitting: <u>If we have too many Features</u>, the learned hypothesis may fit the training set very well

$\left(J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \simeq 0 \right)$, but

Fail to generalize to new examples (predict prices on new examples).

→ Then, Overfitting occures when we have too many features

For the First example (housing prices)

x_1 = size of house

x_2 = no. of bedrooms

x_3 = no. of floors

x_4 = age of house

x_5 = Kitchen size

⋮

x_{100}

Two Ways To Solve This Problem.

1. Reduce Number Of Features.
   - Manually select which Features to Keep (most important
   - Using Model Selection Algorithm.

2. Regularization
   - Keep all the Features, but reduce magnitude (values of parameters θ_j).

For housing price Example.

→ Just Right $\Rightarrow h_\theta(x) = \theta_0 + \theta_1 X + \theta_2 X^2$

→ OverFitting $\Rightarrow h_\theta(x) = \theta_0 + \theta_1 X + \theta_2 X^2 + \theta_3 X^3 + \theta_4 X^4$

to solve the problem of overFitting using regularization we reduce values of θ_3 and θ_4 so $\theta_3 X^3 + \theta_4 X^4 \simeq 0$.
So we return to Just Right (Best Fitting way).

Note: we reduce this parameters θ_j during iteration This mean that we use them not the early number of iteration and start to reduce them step by step.

- Works well when we have alot of Features, each of which Contributes a bit to predicting y.

→ we will Study Regularization in details.

→ Cost Function.

- Regularization.

  Small values for parameters $\theta_0, \theta_1, \dots \theta_n$

  - "Simpler" hypothesis
  - less prone to overfitting

Features : $x_1, x_2 \dots x_n$

Parameters : $\theta_0, \theta_1, \dots \theta_n$

Cost Function $J(\theta)$

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \underbrace{\lambda \sum_{j=1}^{n} \theta_j^2}\right]$$

goal : $\min J(\theta)$

* The extra regularization term at the end is used to shrink every single parameter θ

$$\left[\lambda \sum_{j=1}^{n} \theta_j^2\right]$$

- λ is the regularization parameter that controls a trade of between 2 different goals :
  - → The First goal objective, is that we would like to fit the training data well.
  - → The Second goal is, we want to keep the parameters small and therefore keeping the hypothesis relatively simple to avoid overfitting.

The Value of λ mustn't be very large or very small the best value of λ is from 0 to 0.99

→ Regularized linear regression.

- Cost Function

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

- Gradient descent

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Repeat {

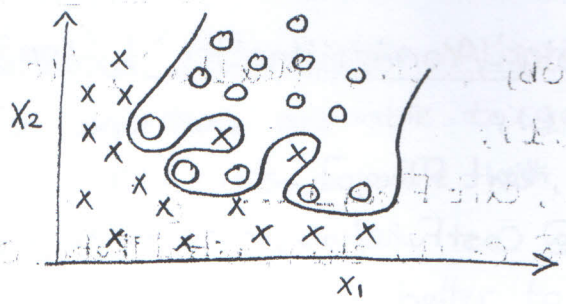
$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

$$(j = 1, 2, 3, \cdots n)$$

}

→ Regularized Logistic regression



$$h_\theta(x) = g(\theta_0 + \theta_1 X_1 + \theta_2 X_1^2 + \theta_3 X_1^2 X_2 + \theta_4 X_1^2 X_2^2 + \theta_5 X_1^2 X_2^2 + \cdots)$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

. Cost Function

$$J(\theta) = \left[\frac{-1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right)\right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

. Gradient descent

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

$(j = 1, 2, 3, \ldots, n)$

}

- Advanced optimization.

Code

```
options = optimset ('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros (n, 1);
[optTheta, functionVal, exitFlag] ...
        = fminunc (@costFunction, initialTheta, options)

function [jVal, gradient] = costFunction (Theta)

    jVal = [ code to compute J(Θ) ];
```

$$J(\Theta) = \left[-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\Theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2$$

```
    gradient (1) = [ code to compute ∂/∂Θ0 J(Θ) ];
```

$$\frac{\partial}{\partial\Theta_0} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})\,x_0^{(i)}$$

```
    gradient (2) = [ code to compute ∂/∂Θ1 J(Θ) ];
```

$$\frac{\partial}{\partial\Theta_1} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})\,x_1^{(i)} - \frac{\lambda}{m}\Theta_1$$

```
    .
    .
    .

    gradient (n+1) = [ code to compute ∂/∂n J(Θ) ];
```